# STORM: A Secure Overlay for P2P Reputation Management

Aina Ravoaja, Emmanuelle Anceaume

# STORM: A Secure Overlay for P2P Reputation Management

Aina Ravoaja
IRISA
Campus Universitaire de Beaulieu
Rennes, France
aravoaja@irisa.fr

Emmanuelle Anceaume
IRISA
Campus Universitaire de Beaulieu
Rennes France
anceaume@irisa.fr

## Abstract

*A fundamental problem that confronts decentralized reputation systems is the design of efficient, secure and incentive-compatible mechanisms to gather trust information despite malicious peers, and particularly collusion. This paper presents STORM (for* Secure sTructured Overlay for Reputation Management*), a P2P protocol which addresses this issue. STORM provides support for gathering peers reputation information safely by self-organizing peers into clusters of peers of common interests. To mitigate the effect of malicious peers, STORM relies on a randomized decision algorithm aiming at securing the routing table maintenance, and on constrained redundant routing focusing on securing information lookup. We show STORM efficiency and robustness through a formal analysis. Specifically, we prove a lower bound on the number of malicious peers that can be inserted in routing tables and we show that our solution is very close to optimal. Finally, we show that our constrained redundant routing approach succeeds at countering collusive behavior.*

## 1 Introduction

Enabling e-commerce in decentralized peer-to-peer systems relies on the consumer confidence in online service providers. Reputation systems have been proven effective at encouraging trust among peers that usually do not know each other [13]. They enable consumers to decide whether a future interaction with a target entity is conceivable or not by collecting, aggregating and ranking opinions about the past behavior of that target peer. Establishing trust in decentralized P2P systems is a fundamental problem, essentially because peers are autonomous, rational, and can be strategic. Peers are rational because they wish to maximize their utility (either by taking advantage of the efficiency of the reputation mechanism, or by maximizing their own reputation, or by reducing their contribution in the reputation estimation effort to save their resource). Peers can be strategic because they can devise complex strategies to subvert the system (e.g. they can manipulate the routing tables or intercept requests to subvert the system). This impact can be greatly increased when groups of malicious peers coordinate their actions to achieve a common goal. Collusion of malicious peers may cause severe damages on the whole system online market. By sending unfair opinions about target peers behaviour, they can abusively inflate the reputation of that peers, and push users to be involved in fraudulent transactions. On the other hand, harming competitors' reputation may discourage further interactions with that competitors leading to unfair market. Thus reputation mechanisms must be robust to a variety of severe attacks, coordinated or not.

**Motivations.** In a prior work [2], we proposed a solution to the robust reputation problem. Essentially this problem aims at evaluating peers reputation despite free-riding and dishonest behaviours. In that work, robustness to attack is accomplished through an aggregation technique in which collected first-hand feedback is weighted by a credibility factor locally computed by the requesting peer, while incentive for participation is implemented through a fair differential service mechanism relying on peers' level of participation and peers' credibility. We showed within modest churn the presence of a high fraction of malicious peers does not prevent a correct peer from accurately evaluating the reputation value of a target peer. However, this is achieved at the expense of a non negligible number of messages mainly due to the way feedback are collected. Indeed, to face malicious peers, feedback is gathered from peers randomly selected within the system which clearly makes the efficiency of this crawling technique highly reliant on the way the graph of witnesses is constructed. Finding the right set of witnesses in an efficient way is a challenging issue since the reputation value depends on the feedback provided by these peers.

**Contributions.** In the present work, we present a cost effective solution to this problem by relying on two schemes: first, to increase the likelihood of collecting a large amount of feedback from the right set of witnesses, secure gathering techniques are identified. Briefly, these techniques aim at guaranteeing that feedback is collected only from a cluster of peers sharing a similar interest for the target service provider, and that bribes and collusion do not interfere by constraining the gathering scheme. STORM is designed to gather information on the reputation of a peer in $O(\log N)$ steps, with $N$ the number of peers. Complementary to this scheme, secure routing tables maintenance and secure routing techniques are provided. These techniques ensure first that routing tables cannot be attacked by malicious peers by keeping the distribution of malicious peers uniform. This is achieved by a *randomised decision* algorithm, which prevents malicious peers from strategizing to get inserted in routing tables. We show that this algorithm eventually minimises the expected number of malicious peers in routing tables. Second, to prevent malicious peers from dropping or delivering a feedback request to their collusion group instead of a legitimate peer, STORM uses *constrained redundant routing*. This scheme guarantees that each request is successful with high probability if the fraction of colluders in a cluster does not exceed $1 - \left(\frac{1}{2(1-c')^{(\log_2 N)/2}}\right)^{\frac{2}{\log_2 N'}}$, with $N$ the size of the system, $N'$ the size of the cluster, and $c'$, the expected number of malicious peers in routing tables. To summarise, the contribution of this paper is a scalable protocol for efficient and secure information gathering in a system with high churn and presence of colluding peers.

**Roadmap.** The paper is organised as follows: Section 2 presents related work, Section 3 presents the model and problem definition, Section 4 presents the basic protocol, Section 5 addresses the security issues, and finally, Section 6 concludes. For space reasons pseudo-codes of some procedures and involved proofs of correctness are presented in the full version of the paper [**?**].

## 2 Related Work

This section analyses previous work on storage reputation information trustworthily despite collusion and on secure routing. Aberer and Despotovic [1] propose a reputation mechanism in which trust information, actually complaint information, is stored in P-Grid, a distributed hash table-based overlay distributed over the system. Their mechanism is made robust by guaranteeing that trust information is replicated at different peers, and thus can be accessed despite malicious entities. The efficiency of their approach relies on peers propensity to fully cooperate by storing trust information locally and by forwarding requests to feed the P-Grid overlay. Additionally, as for most of the DHT-based approaches, peers have to store data they are not concerned with, and thus for different reasons may discard them. This is to prevent such issue that in STORM, trust information are stored at peers sharing similar interest with such information. Simulations based on P-Grid have shown that even sophisticated metric trust are vulnerable to collusion and worse may render the system less efficient than without these metrics [17]. More generally, Zhang et al. [19] have shown that eigenvector-based reputation systems such as PageRank [11] and HITS [10] are very sensitive to collusion essentially because they rely on the notion of transitive trust which can be easily manipulated. To face this issue, presence of pre-trusted peers [9], hash-function applied on long-lived peers [8], and reciprocative-based incentive techniques [7] are proposed. While these approaches show heir effectiveness against different undesirable scenarios, the former one requires the presence of trusted peers known by all the peers in the system, while the two last one can only tolerate modest churn.

Regarding the secure routing problem, Castro et al. [3] are among the first ones to focus on secure routing. They propose the *redundant routing* approach. Essentially, their technique consists in for each request, to send multiple copies of that request over independent routes so that at least one copy almost surely reaches a legitimate recipient. We extend their approach to cope with colluders by constraining the result of a query, which guarantees to reach the legitimate receipeient with high probability. Sit and Morris [15] take advantage of the fact that distance between the sender and the legitimate destination should decrease at each step of the lookup. By making the sender observe his query progress, he can detect re-routing toward malicious destinations, however their implementation doubles the optimal cost of a lookup.

## 3 Model and Problem Definition

We adopt the model proposed in [2]. We consider a peer-to-peer system populated with at most $N$ peers. Some peers called *service providers* repeatedly offer the same service to interested peers. The effort exerted by a service provider determines its Quality of Service (QoS). We assume that a server 's effort is the same for all the peers that solicit him and takes its value within the [0,1]. After each interaction with server $s$, each peer has an imperfect observation of the effort exerted by $s$. Peers may have different tastes about $s$ QoS, but basically, these observations are closely distributed around $s$'s effort. Estimation of the expected behaviour of a server $s$ is based on its recent past behaviour, that is, its recent interactions with the peers of the system. Every time a peer desires to interact with a service provider

$s$, it asks for feedback from peers that have recently interacted with $s$. In the following, a *witness* denotes a peer having interacted with a given service provider.

Some peers try to manipulate the system by exhibiting undesirable behaviours. Such peers are called *malicious*. Malicious peers can drop messages, forward requests to illegitimate peers, send dishonest feedback or no feedback at all. Note that malicious peers are rational, that is, they try to maximise their utility within the system. Thus they follow STORM protocol when it is in their interest to do so. However, they can be strategic, that is, they can elaborate strategies to subvert the system. Malicious peers may act independently or may be part of a *collusion group*. A member of a collusion group is called *colluder*. A peer which always follows STORM protocol is said to be *correct*. We assume that there exists at most a fraction $f(0 \leq f < 1)$ of malicious peers in the whole system. The set of malicious peers is partitioned into independent disjoint collusion groups with size bounded by $cN(1/N \leq c \leq f)$. When $c = f$, all malicious peers collude with each other to cause the most damage to the system [3]. In this work, we do not consider sybil attacks [6], that is peers that pollute the system by creating numerous fake identifiers. We suppose that STORM uses some external technique to counter this problem and leave this issue for future work. We use cryptographic techniques to prevent a malicious peer from observing or modifying a message physically sent between any two correct peers. However a malicious peer has complete control over messages it receives, i.e., it can drop them or forward them to illegitimate peers. Finally, we assume that every peer in the P2P overlay has static IP address at which it can be contacted. Note that P2P overlays can be extended to address dynamic IP addresses [5].

**Specification of the Secure Gathering Problem.** A reputation mechanism collects, aggregates and rates feedback. Collecting honest feedback in a dynamic environment in which a non null fraction of malicious peers can collectively collude to subvert the system is the problem we address in this work. A solution to this problem should satisfy the following property:

**Property 1 (Secure feedback collect).** *If a peer $p$ requests $w$ opinions regarding the service provided by $s$, then $p$ receives $w$ such opinions with high probability, and each one is sent by a correct witness with probability $1 - f'$.* [1]

# 4 STORM Protocol

STORM is a scalable protocol for efficiently gathering numerous and honest feedback to evaluate the reputation of service providers. The principles that led to STORM are the following ones: first, to increase the likelihood of locating sufficiently enough feedback, all the peers having interacted with a target server (witnesses) should dynamically self-organise into clusters with respect to that target server. In the following, *cluster*$_s$ denotes the group of peers having interacted with service provider $s$. Second, to efficiently retrieve these opinions in a large scale system, each such cluster should be reachable in a logarithmic number of hops. Finally, to prevent collusive peers from subverting the gathering scheme (either by re-routing, by dropping feedback, or by flooding the system with dishonest feedback), collusion formation should be prevented. All these tenets have been integrated in STORM and are now presented.

This section describes the STORM protocol, namely, location of service providers and witnesses, insertion of new peers and departures of existing ones. Prior to this description, we recall some background on DHT-based protocols, by focusing on Chord. Indeed, we take advantage of the attractive features of Chord (i.e., its scalable key location and dynamic join and leave operations) to adapt them to the secure gathering problem. Note that other DHT-based protocols such as Pastry [14] or CAN [12] could have been adapted too, however, the choice of Chord was motivated by security considerations as discussed in Section 5.

## 4.1 Background

### 4.1.1 DHT-based Protocols

DHT-based protocols [16, 14, 12] provide substrates for the construction of large-scale, decentralised applications. They allow applications to locate an object in a probabilistic bounded small number of hops while requiring per-peer routing tables with only a small number of entries. These protocols are scalable, fault-tolerant and provide effective load balancing. In a structured overlay, peers are assigned unique random identifiers, *nodeIds*, from a large *id space*. Application-specific objects are assigned unique identifiers, called *keys*, selected from the same id space. Each key is mapped by the overlay to a unique peer, called the key's *root*. The routing protocol routes messages with a given key to its associate key's root. To route messages efficiently, each peer maintains a *routing table* with nodeIds of other peers and their associated IP address.

### 4.1.2 Chord Protocol

In Chord, peers are uniformly assigned nodeIds from a circular $m$-bits id space through a consistent hashing function, with $m = \log_2 N$, and $N$ the size of the system. Each Chord peer maintains a routing table, called *finger table*, consisting of up to $m$ pointers to other peers. The $i$th pointer of peer (of nodeId) $n$ refers to the peer with the smallest nodeId

[1] $f'$ is explicited in Section 5.

clockwise from $n + 2^{i-1}$. Each Chord key is mapped to the peer with the closest preceding nodeId to the key. To find a particular key, a request is forwarded in clockwise direction to the peer in the routing table with the closest preceding nodeId to the key. Thus, the expected number of routing hops in Chord is $(\log_2 N)/2$. A newly joining peer $p$ initialises its fingers, and notifies existing peers, which in turn updates their fingers to reflect $p$'s arrival. Chord handles abrupt peers departure through a periodic invocation of a stabilisation protocol which exchanges messages among a small number of peers. A complete description of Chord is presented in [3].

## 4.2 STORM Overview

As previously said, STORM adapts Chord to gather opinions regarding the service provided by servers. Briefly, the very notions of keys and peers are at the roots of the adaptation we propose. As previously mentioned, Chord assigns keys to peers. That is application resources are physically assigned to peers. In STORM, the notion of cluster of witnesses acts as a substitute for the notion of application resource, while the notion of (physical) peer is replaced by the one of (logical) label or key. Specifically, each cluster $cluster_s$ is assigned as a whole to a *key* which is obtained by hashing the IP address of the service provider $s$. Keys are ordered on a *key circle* modulo $2^m$. Note that $m$ must be large enough to make the probability of two service providers IP addresses hashing to the same key negligible. In the rest of the paper, key circle will refer to the *backbone ring*, and the term "service provider" will refer to both the service provider and its key under the consistent hash function. Let us now focus on clusters structure. A consistent hash function assigns witnesses of a given cluster *identifiers* ordered on a *identifier circle* modulo $2^n$.[2] The identifier circle is referred in the following as the *witnesses ring*. Note that a peer may belong to $x$ witnesses rings if it has interacted with $x$ service providers. Figure 1) illustrates the backbone ring on which five witnesses rings $W_1$, $W_2$, $W_3$, $W_5$, $W_6$ sit, with $W_s$ the witnesses ring including witnesses having interacted with server $s$. For clarity purpose, we focus on witnesses ring $W_1$, $W_2$, and $W_3$, and consider only peer $p_1$'s tables. Each witness $p \in W_s$ maintains two tables: a *routing table*, denoted $Routing_p^s$, to maintain the inter-ring connectivity, and a *witnesses table*, denoted $Witnesses_p^s$, to maintain the inner-ring connectivity (both tables are described hereafter). Beyond such tables, each witness stores the opinion it has about the experiences it has had with the given service provider. Note that game theoretic results and empirical studies on eBay show that only recent ratings are meaningful [4]. Thus a witness stores only its latest observations.

Note that in most of the reputation systems based on DHT overlays [9, 1] the opinions of peers are located at remote peers. Specifically, the feedback that peer $p$ has on some target server $s$ is not locally stored at $p$ but rather is owned by some remote peer $p'$ whose location is given by some consistent hash function, for instance by hashing $s$'s Id to a key. Generally, $p'$ has no reason to be interested in the feedback related to $s$, while the cost incurred by storing such information is not negligible. As a consequence, it may be a clear disincentive to actively and honestly participate to the reputation process. This is the reason why the alternative approach has been adopted in STORM.

### 4.2.1 Witnesses Table

As aforementioned, witnesses tables ensure the inner connectivity of a witness ring. For scalability reasons, we do not require each witness to know about every other witness in the witnesses ring. Rather, a witness points to only a small number of other witnesses. Thus, similarly to Chord's *finger table*, $p$'s witnesses table contains at most $m$ entries, such that the $i^{th}$ entry points to the first witness $p'$ that succeeds $p$ by $2^{i-1}$ on the ring (that is $p'$ is the first peer that succeeds $(p + 2^{i-1}) \bmod 2^m$, with $1 \leq i \leq m$). Entry $i$ includes both $p'$'s *identifier* and $p'$'s IP address. Witness $p'$ is indifferently called the $i^{th}$ witness of $p$, or the $i^{th}$ *succeeding witness* of $p$. By symmetry, $p$ is a *preceding witness* of $p'$. From Chord properties, each witness maintains information about only $O(\log N)$ other witnesses, and with high probability the number of witnesses that must be contacted to find a particular witness in a witnesses ring is $O(\log N)$ (Theorem IV.2 of [16]).

### 4.2.2 Routing Table.

Routing tables ensure the connectivity between the different witnesses rings. As for witnesses rings, only a small amount of routing information can be maintained by each witness. Thus $p$'s routing table contains at most $m$ entries, such that the $i^{th}$ entry refers to the first witnesses ring $\mathcal{W}_{s'}$ that succeeds $\mathcal{W}_s$ by at least $2^{i-1}$ on the backbone ring. In the sequel $\mathcal{W}_{s'}$ is called the $i^{th}$ successor ring of $W_s$, or simply a *successor* of $W_s$. By symmetry, $\mathcal{W}_s$ is a *predecessor* of $\mathcal{W}_{s'}$. From here on, we denote by $E_p[i]$ the $i$th entry of the routing table of $p$. Each entry $E_p[i]$ contains a set of witnesses' identifiers associated to their IP address, such that each of those witnesses belongs to $\mathcal{W}_{s'}$. We have $|E_p[i]| \leq d$ for all $i$, with $d$ an application parameter. Referring to Figure 1, the routing table $Routing_1^2$ of peer $p_1$ contains 3 entries $E_1[1]$, $E_1[2]$, and $E_1[3]$ referring respectively to $W_3$, $W_5$, and $W_6$. The first entry $E_1[1]$ of $Routing_1^2$ refers to the witnesses ring $W_3$, and points to witnesses $p_2$ and $p_5$. As will be described in Section 5, this set

[2] In practice $m = n$ since each witness may potentially be a service provider.

is introduced first for security reasons (it aims at preventing collusion formation), and second for fault-tolerance considerations. Assuming that each peer fails independently with probability $p_f$, the probability that all $d$ witnesses fail simultaneously is only $p_f^d$. Note that $d$ is independent of the size of the system $N$, and thus the amount of information maintained by a peer in its routing table is in $O(\log N)$. Putting things together, the amount of information needed to maintain both tables is $O(\log N)$.

**Figure 1. A STORM architecture with $m = 3$ and $d = 2$**

### 4.3 Routing Procedure

We now describe how a witnesses ring is located in STORM. Basically, the routing procedure of STORM follows the lines of Chord but applied to witnesses rings. The first difference lies in the fact that unlike in Chord where a finger table entry contains the identifier of a single peer, in STORM a routing table contains a *set* of peers. Then at each hop of a STORM message routing procedure, the next peer is chosen from the relevant routing table entry at random. This random choice is motivated by load balancing and, more importantly, by security considerations (see Section 5.2). The second difference is that a STORM message keeps track of the first encountered peer $pred$ that refers to the target witnesses ring. Finding such a peer is useful for the update procedure (Section 4.4), for the secure insertion procedure (Section 5.1), and for the secure routing procedure (Section 5.2).

When a peer $p$ sends a request for the witnesses ring $\mathcal{W}_s$ of service provider $s$, $p$ executes the `find_witness()` procedure. This procedure returns a tuple ($pred$,$target$), with $pred$ a peer that refers to $\mathcal{W}_s$, and $target$ a peer in $\mathcal{W}_s$. `find_witness()` consists of the following steps. If the routing table of $p$ contains a non empty entry $E_p[i]$ (that refers to $\mathcal{W}_s$) then `find_witness()` ends and returns $(p, p')$, with $p'$ a peer chosen at random from $E_p[i]$. Otherwise, $p$ searches in its routing table for the entry $E_p[i]$ that refers to the witnesses ring $\mathcal{W}_{s'}$ whose key $s'$ most immediately precedes $s$. Then $p$ picks a random peer $p'$ from $E_p[i]$, which in turn invokes `find_witness()`. Note that if key $s$ of $\mathcal{W}_s$ falls between key $s'$ of $W_{s'}$ and key $s''$ of the witnesses ring $\mathcal{W}_{s''}$ that immediately succeeds $\mathcal{W}_{s'}$, then it means that $\mathcal{W}_s$ does not exist and `find_witness()` returns ($p'$, $null$).

Locating a witness within the witnesses ring is done exactly as previously, with the difference that witnesses tables contain only one pointer per entry.

As the backbone ring is placed on a Chord-like ring, searching for a witnesses ring costs $O(\log N)$ hops.

```
   /* peer p wishes to find W_s                                   */
 1 p.find_witness(s) {
 2     E_p = entry of p that refers to the closest ring W_s' to W_s;
 3     p' = choose a random peer from E_p;
 4     q = p;
 5     while( s ∉ [s'; s''] ){
 6         q = p';
 7         E_q = entry of p that refers to the closest ring W_s'' to W_s';
 8         p' = choose a random peer from E_q;
 9     }
10     if ( s = s' ){
11         result.pred = q;
12         result.target = p';
13     }else{
14         result.pred = p';
15         result.target = null;
16     }
17     return result;
18 }
```

**Algorithm 1**: Routing procedure

### 4.4 Handling Witnesses Arrivals and Removals

This section presents how new witnesses are inserted on the STORM rings, and how departures (voluntary or not) of existing witnesses are handled.

#### 4.4.1 Insertion Operation

When an interaction between service provider $s$ and peer $p$ is completed, peer $p$ may want to join STORM as witness to actively participate to the estimation of server $s$'s reputation. In the sequel, we assume that $p$ knows at least one peer already inserted in STORM. A bootstrap peer can be obtained using for example a set of dedicated servers. Peer $p$ issues an insertion request to the bootstrap peer in order to be inserted in $\mathcal{W}_s$ (invocation of a "join" procedure). The "join" procedure first locates the witnesses ring $\mathcal{W}_s$ on the backbone ring using the `find_witness()` procedure described hereabove. As previously explained, this procedure returns a tuple $(pred, target)$, with $pred$ a member of a witnesses ring $\mathcal{W}_{s'}$ that points to $target$, with $target \in \mathcal{W}_s$. Two cases have to be considered:

**case a)** $target = null$**, i.e.,** $\mathcal{W}_s$ **does not exist**. In that case, $\mathcal{W}_s$ has to be created and this creation has to be reflected in all the witnesses rings that precede $\mathcal{W}_s$ on the backbone ring. Creation of $\mathcal{W}_s$ is done by initialising $p$'s routing table (by successively contacting the successors of *preds*). $p$'s witness table is not initialised at this stage since $p$ is the only member of $\mathcal{W}_s$.

Creation of $\mathcal{W}_s$ is reflected in the other routing tables by invoking a procedure which updates the routing tables of peers concerned in $p$'s arrival. Briefly, this procedure

consists in contacting all the witnesses belonging to all the witnesses rings $\mathcal{W}_{s'}$ that precede $\mathcal{W}_s$. Each witness $p'$ in $\mathcal{W}_{s'}$ inserts $\mathcal{W}_s$ as a new entry $i$ in its routing table, and makes this entry point to $p$, i.e., $E_{p'}[i] = p$. Note that this costly procedure is applied only at the creation of a witnesses ring. Once a witnesses ring exists, the insertion of a new witness in that ring is reflected in at most one routing table entry (see below). Note also that once a witness has been inserted in a witnesses ring, his next interactions with the same service provider are transparent for STORM: feedback is locally updated if necessary, but no insertion operation has to be invoked as long as he does not leave that witnesses ring. Malicious peers may try to be inserted several times in the same witnesses ring to subvert the system, however the insertion procedure guarantees that once a peer exists in the ring it cannot be inserted a second time.

**case b)** $target \neq null$**, i.e.,** $\mathcal{W}_s$ **exists**. In that case, $p$'s insertion in $\mathcal{W}_s$ needs to be reflected at some of the members of $\mathcal{W}_s$ and to be possibly incorporated in *pred*'s routing table. Specifically, $p$'s insertion is achieved by first locating his position on $\mathcal{W}_s$, and then by asking $p$'s preceding witnesses to update their witnesses tables to incorporate $p$ as witness (recall that witness $p'$ precedes $p$ if it exists $i$ such that $p$ is the first peer that succeeds $(p' + 2^{i-1}) \bmod 2^m$, with $1 \leq i \leq m$). Then $p$ initialises his own witnesses table by successively contacting its succeeding witnesses. This procedure follows the lines of the "join" procedure of Chord.

On the other hand, and contrary to case a) only one routing table is susceptible to incorporate $p$'s identifier. This routing table is $pred$'s one, and $p$'s incorporation is conditioned by the sequence of insertion requests previously received by $pred$. Indeed, by constraining peers' insertion in routing tables, we prevent malicious peers from devising strategies (collective or not) to infect routing tables entries. Consequently we decrease the likelihood that such peers get control over both the witnesses ring they point to (and thus the danger that they illegitimately reply to requests addressed to other members of those witnesses rings), and the backbone ring (and thus the risk that they intercept requests they should forward). The *randomised decision* algorithm responsible for that insertion is presented in Section 5.

#### 4.4.2 Departure and Failure Operation

By construction STORM does not impose any constraints on feedback availability. Indeed, since witnesses are responsible for their own feedback, feedback of failed left witnesses does not have to be kept by other peers. Thus, the only property that needs to be maintained by STORM is to guarantee that peers' departure (volunteer or not) does not compromise connectivity on the backbone and within witnesses rings. Regarding connectivity among the witnesses rings, each peer periodically sends alive messages to each of his successors of his routing table. If a peer notices that one of his successors has left then it removes him from his routing table. Recall that each routing table entry contains up to $d$ witnesses pointing to a same witnesses ring, and thus up to $d$ concurrent failures have to occur to damage witnesses rings inter-connectivity, and this case happens only with probability $p_f^d$. Regarding connectivity within witnesses rings, we adopt the solution proposed in Chord. Their solution consists in maintaining at each witness $p$ a successor list containing $p$'s $b$ immediate successor. If a $p$'s immediate successor does not respond, $p$ can substitute the second entry in its successor list. As for routing tables entries, up to $b$ concurrent failures have to occur to damage connectivity within witnesses rings, which happens with only a probability $p_f^b$.

## 5 Security Issues

In this section we focus on the security aspects of STORM. We argue that DHT-based overlay for reputation systems should satisfy the following two properties:

- Routing tables should be insensitive to collusion. That is, if before a table update procedure the probability that a link points to a malicious peer is $f'$, then after the update this probability should not exceed $f'$.
- Routing should be efficient. That is, a request should reach a legitimate destination (i.e. should neither be dropped, nor derouted) with high probability.

Probability $f'$ corresponds to the maximal fraction of malicious peers that can be inserted in a witnesses ring that is $f\frac{N}{|\mathcal{W}_s|} < f'$ for all service providers $s$.

In the sequel we propose strategies that enjoy both properties, and show that their combination allow to build a secure feedback gathering procedure.

Before describing our contribution regarding secure routing table maintenance and secure routing, we present arguments which guarantee that peers cannot choose their own identifiers or identifiers of other peers. These arguments are very similar to Castro et al's ones [3]. First, by safely assigning keys and identifiers through a trusted Certification Authority (CA) (central or installed as part of the peer software itself), peers cannot practically forge a key or an identifier. Second, from the consistent hash function property, keys and identifiers are distributed uniformly at random over the backbone ring and over each witnesses ring. Moreover, as keys and identifiers are assigned through a cryptographic hash function, malicious peers can arduously know which IP address has generated a particular key

or identifier. Thus, they cannot target a particular point on the key or identifier space. Finally, from Chord properties, strong constraints on the keys and identifiers positions on the rings are imposed, i.e., a peer has to be the closest to some fixed point on the ring. Thus a peer cannot manipulate the system to be inserted close to a target peer, this target being either another malicious peer to form a collusion, or a correct peer harmed by the manipulating peer.

### 5.1 Secure Insertion in Routing Tables

The aforementioned arguments are sufficient to prevent malicious peers from choosing their position in witnesses rings. However they cannot prevent them from choosing the witnesses ring in which they will be inserted. Indeed, the only condition required to be inserted in a witnesses ring $\mathcal{W}_s$ is to claim a recent interaction with service provider $s$. On the other hand, additional mechanisms can be designed to prevent malicious peers from devising strategies (collective or not) to infect routing tables entries. We address this issue by randomising the insertion process. Intuitively, colluders can more easily draw a successful adversarial strategy from a deterministic algorithm than from a randomised one. For example, colluders may easily subvert a First-In-First-Out (FIFO) insertion algorithm by executing periodic rapid-fire insertion requests. In contrast, a colluder can hardly make up a successful strategy if the inserted peers are chosen randomly among the requesting peers. We show that regardless of the adversarial strategy colluders employ, the randomised decision algorithm we propose guarantees that the expected number of colluders in each routing table is eventually minimal.

Specifically, suppose that among a sequence of $t$ insertion requests, colluders can send at most $x$ requests, with $x < t$, and that colluders can strategize on the order of these requests. For instance, they can send all the $x$ insertion requests at the beginning of the sequence, or they can spread the $x$ requests over the whole sequence. Note that $x/t \leq f'$. Our algorithm guarantess that the expected number of malicious peers inserted in a routing table entry is at most $(x/t)d$. For $t = 30$, $x = 10$, and $d = 10$, if a FIFO insertion policy were used, colluders would be able to infect $100\%$ of the peers in the routing table entry by sending rapid-fire insertion requests. Using our randomized algorithm, only $33\%$ of these peers would be malicious. Note that to prevent a malicious peer $p$ already inserted in a witnesses ring $\mathcal{W}_s$ to send new insertion requests in order to infect the routing table of a peer $pred$ that refers to $\mathcal{W}_s$, $pred$ checks whether $p$ is not already in $\mathcal{W}_s$ before executing the randomized insertion algorithm.

In the following, an insertion request originating from a colluder is called a *malicious insertion request*. Now, let $p$ be a peer issuing a insertion request to be in $\mathcal{W}_s$. Let *pred* be the peer belonging to $\mathcal{W}_{s'}$ such that $\mathcal{W}_{s'}$ precedes $\mathcal{W}_s$, and such that the $i$th routing table entry of *pred*, $E_{pred}[i]$, contains up to $d$ witnesses belonging to $\mathcal{W}_s$. Finally, let *transit*$_i$ be a list containing the identifiers of the $t$ last peers that have solicited *pred* to be inserted in $\mathcal{W}_s$. Upon receipt of $p$'s insertion request, *pred* runs the *randomised decision* algorithm which consists of the following five steps:

1. if *transit*$_i$ is full, remove the oldest peer from it,
2. insert $p$ in *transit*$_i$,
3. pick at random a peer $p'$ from *transit*$_i$,
4. if $E_{pred}[i]$ is full then drop a random a peer from $E_{pred}[i]$,
5. insert peer $p'$ in $E_{pred}[i]$.

Recall that this algorithm is executed only if $\mathcal{W}_s$ exists (see Section 4.4). Before showing that this algorithm guarantees that the expected number of malicious peers is eventually equal to $(x/t)d$, we first show that the algorithm is vulnerable to malicious peers during the STORM creation. By construction, the first $i$, with $i < t$, peers issuing an insertion request are inserted in $E_{pred}[i]$ with probability $1/i$. After that, the probability of insertion of a peer in $E_{pred}[i]$ is always equal to $1/t$. Clearly, as colluders generally try to maximize the probability of being inserted in a routing table entry, the $t$ first insertions constitute the most vulnerable phase of the randomized decision algorithm. Note that a possible alternative would be to wait for the $t$ first insertion requests before inserting the first peer in $E_{pred}[i]$. The drawback of this strategy would be to prevent requesting peers from getting any feedback for service provider $s$ during this initial period of time. While this vulnerability exists, the routing table entry is eventually "cleaned up" by our randomized algorithm as now demonstrated.

**Analysis.** In the following we show first that the expected number of malicious peers in each routing table is eventually at most equal to $(x/t)d$ (Theorem 1), and second that for any randomized algorithm, there exists adversarial strategies for which the expected number of malicious peers is equal to $(x/t)d$ (Theorem 2).

**Theorem 1.** *Using the randomized decision algorithm, the expected number of malicious peers in a routing table entry is eventually at most equal to* $(x/t)d$.

**Theorem 2.** *For any randomized algorithm, there exists a sequence of* $t$ *insertions such that the expected number of malicious peers in a routing table entry is at least* $(x/t)d$.

The randomized decision algorithm eventually bounds the expected number of malicious peers in each routing table entry to $(x/t)d$, which is the best result that can be

achieved. Note however that this mechanism cannot be used to minimize the number of malicious peers in witnesses tables. Indeed, by excluding some peers from witnesses tables, their potential feedback would not be available, which would degrade the efficiency of the reputation estimation.

### 5.2 Secure Routing

The previously described randomized decision algorithm prevents the formation of collusion. Even if this schema limits the likelihood of requests being dropped or being routed toward undesirable peers, current structured p2p overlays provide best-effort routing. The probability of reaching a legitimate recipient is small for the simple reason that only one malicious peer on the path is sufficient to prevent a request from being delivered to its destination. To illustrate this problem, consider a system of $N$ peers and a target witnesses ring $\mathcal{W}_s$ of size $N' \leq N$. By assumption the fraction of malicious peer in $\mathcal{W}_s$ and thus in each witnesses table is at most $f'$. We have shown (see Theorem 2) that regardless of the adverserial strategy, the expected number of malicious peers in each entry of each routing table is equal to $(x/t)d$ which is the best result that can be achieved. From the routing procedure (Section 4.3), the peer at which a request received by peer $q$ is forwarded is chosen at random from the appropriate entry of $q$'s routing table. Thus the probability of choosing a malicious peer at each hop is equal to $c' = x/t$. Moreover, the expected number of hops needed to reach $\mathcal{W}_s$ is equal to $(\log_2 N)/2$ and the maximum expected number of hops needed to reach a target peer in $\mathcal{W}_s$ is equal to $(\log_2 N')/2$. Therefore from above, the probability that a request reaches a legitimate destination is given by the following equation:

$$(1 - c')^{(\log_2 N)/2}(1 - f')^{(\log_2 N')/2}.$$

Note that this legitimate destination is correct with probability $1 - f'$. Figure 2 illustrates this result by plotting the probability of a successful request for different ratio of malicious peers with $N = 100,000$ and $c' = 0.01$. For example, for $N' = 1,000$ we see that 0.08% of malicious peers (that is $80$ malicious peers) in $\mathcal{W}_s$ makes the probability of success of a request for $\mathcal{W}_s$ drop to 60%.

**Figure 2. Probability of success of a request with respect to the fraction of malicious peers in $\mathcal{W}_s$ for $N = 1,000,000$ and $c' = 0.01$.**

To improve the probability of reaching a legitimate destination, Castro et al. [3] propose the *redundant routing* approach. Essentially, for each request, multiple copies of that request are sent over independent routes so that at least one copy almost surely reaches a legitimate recipient. To cope with colluders in $\mathcal{W}_s$ and malicious peers in routing tables, we extend their approach to *constrained redundant routing*. Our approach consists in constraining the result of a request. In our case the constraint concerns the identifier of a witness. Specifically, consider a peer $p$ that wishes to find $w$ witness identifiers in $\mathcal{W}_s$. Then $p$ executes the *constrained redundant routing* algorithm which consists of the following four steps:

1. generate $w$ random numbers $n_k$, $k = 1 \ldots w$ from 0 to $2^m - 1$;
2. send $r$ copies of the feedback request for $\mathcal{W}_s$ over $r$ independent routes;
3. when a peer $pred$ having an entry $E_{pred}[i]$ referring to $\mathcal{W}_s$ receives a feedback request copy, he asks all the $d$ witnesses in $E_{pred}[i]$ to find the closest witness to each $n_k$, $k = 1 \ldots w$;
4. $p$ waits until timer expires;
5. the final result of the feedback request is the $w$ identifiers of the witnesses that have been returned the most often.

The constraint is expressed by Step 3 of the algorithm. With such a constraint, a requester can evaluate the validity of the result and can choose the appropriate one (Step 5). Such an evaluation would be impossible if peers could return arbitrary witnesses identifiers in $\mathcal{W}_s$.

Note that generating $w$ random keys from 0 to $2^m - 1$ and asking for the closest peers to these keys can be interpreted as choosing $w$ peers in the witnesses ring (approximately) randomly. Thus the probability that the closest peer $w_k$ to $n_k$ is correct is $1 - f'$.

By independent routes, we mean that routes are independently *chosen*. Thus sending multiple copies of a request over independent routes can be viewed as making multiple random walks on the set of possible paths starting from the requester to the destination. Recall that a routing table entry may contain up to $d$ identifiers, and that at each hop of a request the subsequent peer is chosen from the appropriate routing table entry at random (see Section 4.3). Thus sending $r$ copies of a request over independent routes simply consists in sending $r$ request messages simultaneously.

Sending each copy of a request to the $d$ peers in each routing table entry that refers to $\mathcal{W}_s$ is motivated by the fact that colluders are more likely to be present in $\mathcal{W}_s$ than in routing tables ($c' \leq f'$). It can be interpreted as checking many possible routes to the $w$ target witnesses to increase the chance to reach them.

**Analysis.** We now show that if the fraction of malicious peers $f' < 1 - \left(\frac{1}{2(1-c')^{(\log_2 N)/2}}\right)^{\frac{2}{\log_2 N'}}$ in $\mathcal{W}_s$, then for each key $n_k$, the most often returned witness $w_k$ is the legitimate destination. Such a request is called a *successful* request. To show this result, we consider that all malicious peers in $\mathcal{W}_s$ collude altogether by claiming that for each key $n_k$, $w'_k$ is the legitimate destination with $w'_k \neq w_k$.

**Theorem 3.** *If $f' < 1 - \left(\frac{1}{2(1-c')^{(\log_2 N)/2}}\right)^{\frac{2}{\log_2 N'}}$, then by setting $r$ and $d$ to large enough values, with high probability a request is successful.*

Table 1 gives some numerical values of the maximum number of colluders that the constrained redundant routing algorithm can handle with respect to the size of $\mathcal{W}_s$ and for $N = 1,000,000$ and $c' = 0.01$.

| Size of $\mathcal{W}_s$ | Maximum size of a collusion group |
|---|---|
| 100 | 16 |
| 1,000 | 115 |
| 10,000 | 876 |
| 1,000,000 | 7078 |

**Table 1. Maximum tolerated size of a collusion group**

Now, for more practical considerations, we seek to determine the probability that a request is satisfied with respect to $r$ and $d$. From the previous analysis, we observe that the number of copies of a request that reaches a legitimate destination follows a binomial law of parameters $p_s$ and $r$, with $p_s = (1-c')^{(\log_2 N)/2}(1-f')^{(\log_2 N')/2}$. Then, the probability of success of a request is given by the following equation:

$$\sum_{k>r/2}^{r} \binom{r}{k} p_s^k (1-p_s)^{r-k}.$$

As illustrated by Figure 3, for $N = 100,000$, $N' = 1,000$, and $c' = 0.01$, with 0.08% of colluders (80 colluders), and for $r = 5$ and $d = 10$, the probability of success of a request increases to 94%.

The expected total number of messages for a constrained redundant routing is $r\left((\log_2 N + wd\log_2 N')/2\right)$. Since $N' \leq N$ and $r$, $d$, and $w$ are fixed, the complexity of the constrained redundant routing algorithm is still $O(\log N)$.

### 5.3 Gathering Feedback

We can finally describe how opinions are gathered from a set of witnesses. When a peer $p$ wishes to get $w$ opinions regarding service provider say $s$, $p$ invokes a "query" function to locate $\mathcal{W}_s$. In case no witness have ever interacted with $s$, this procedure returns a null pointer. Otherwise, $p$ gets back $w$ opinions about $s$. The approach simply consists in executing the secure routing procedure described in Section 5.2 to get $w$ identifiers of witnesses. Then, $p$ directly contacts each witness and requests their recent feedback about $s$.

Since this procedure returns $w$ witnesses of $s$ with high probability (Section 5.2), and since $w$ are chosen at random from $\mathcal{W}_s$ (Section 5.2) and thus is correct with probability $1 - f'$ by assumption, Property 1 of the Secure Feedback Collect is satisfied.

**Figure 3. Probability of success of a request with respect to the number $r$ of copies for each request.**

## 6 Conclusions

We have presented STORM, a scalable protocol for efficient and secure information gathering in a system which high churn and collusive peers. Efficiency comes from two ingredients: The first one is self-organization. By making peers to be adjacent to other peers sharing the same interest, we increase the likelihood of finding relevant information within a small number of number of steps. The second one is Distributed Hash Tables (DHT)-based overlay. By taking advantage of the attractive features of these systems semantic clusters are located in a logarithmic number of hops and high churn is faced through simple and fast leave and join operations. Secure information gathering techniques are proposed. These techniques guarantee that feedback is collected from the legitimate cluster, and bribes and collusion do not interfere the gathering, without having to discover who the undesirable peers are.

## 7 Acknowledgments

We thank the anonymous reviewers for their useful comments that helped improve the paper. This work is partially supported by a grant from brittany region convention # 1306, and the ANR RIAM-011 03 "Solipsis".

## References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-to-peer information system. In *Proceedings of the 10th International ACM Conference on Information and Knowledge Management (CIKM)*, 2001.

[2] E. Anceaume and A. Ravoaja. Incentive-based robust reputation mechanism for p2p services. In *Proceedings of the*

*10th International Conference On Principles Of Distributed Systems*. LNCS 4305, 2006.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[4] C. Dellarocas. How often should reputation mechanisms update a trader's reputation profile. *Information Systems Research*, 2006.

[5] P. Dewan and P. Dasgupta. Pride: Peer-to-peer reputation infrastructure for decentralized environments. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters*, 2004.

[6] J. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[7] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-tto-peer networks. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, 2004.

[8] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Havelaar: A robust and efficient reputation system for active peer-to-peer systems. In *Proceedings of the First Workshop on the Economics of Networked Systems (NetEcon)*, 2006.

[9] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, 2003.

[10] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedinds of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[13] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood. The value of reputation on ebay: A controlled experiment. *Experimental Economics*, 2006.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceeding of the International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[15] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of the First Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[16] I. Stoica, D. Liben-Nowell, R. Morris, D. Karger, F. Dabek, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. 11(1), 2003.

[17] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust in peer-to-peer communities. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2004.

[18] A. Yao. Probabilistic computations: toward a unified measure of complexity. In *Proceedings of the 17th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1977.

[19] H. Zhang, A. Goel, R. Govindan, K. Mason, B. V. Roy, and L. Stefano. Making eigenvector-based reputation systems robust to collusion. *Lecture notes in computer science*, 2004.